Q1

Let q, d, ni and p be the number of coins of quarters, dimes, nickels and pennies respectively.

By value, we know quarters > dime > nickel > penny.

Implementing a greedy algorithm, we select coins of the highest denominations without going over the input total number of cents. We repeat this process until we have 0 cents left to fill.

Explanation →

We start with quarters

$$q = \frac{n}{(\text{value})_{\text{quarter}}}$$

$$n_q = \text{cents left} = n \bmod (\text{value})_q$$

Next, we ~~find~~ move to dime

$$d = \frac{n_q}{(\text{value})_{\text{dime}}}, \quad n_d = \text{cents left} = n_q \bmod (\text{value})_{\text{dime}}$$

Similarly we find n_{ni}

and the remaining cents after that will be equal to 'p'

Prove greedy choice and optimal substructure

* The algorithm always selects the coin with the highest value as long as the resultant remaining cents is not less than the coin value. Then, it moves on to the next highest value coin. Thus, the algorithm is greedy.

* Optimal Substructure.
Assume our algorithm is not optimal.
Let another algo be optimal with N_q quarters, N_d dimes, N_n nickels and N_p pennies
Note if $N_p \geq 5$ means it can be replaced by a nickel
$\therefore N_p < 5$

Similarly, $N_d \leq 2$, and $N_{ni} \leq 1$

Let's say for 14 cents,

Our algorithm will choose one dime and 4 pennies.

Here, any other combination will mean equal / more number of coins but not the highest possible denomination in any other case.

∴ ~~The~~ Any other approach cannot be more greedy than our proposed method and it is valid for any number of cents.

Time Complexity - ~~O(n)~~ O(1)

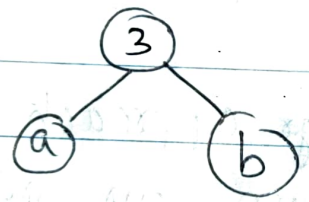
because we simply perform at max four division and four modulus operation to get total number of all coins.

Q2 Bits needed to encode the message "abbcccxxxxyyyzz"

Character	a	b	z	c	y	x
Frequency	1	2	2	3	3	4

We combine characters from smallest frequency to largest

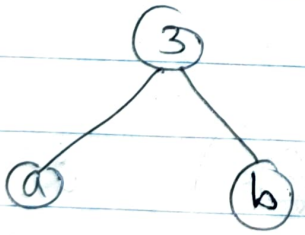
①



2 3 3 4

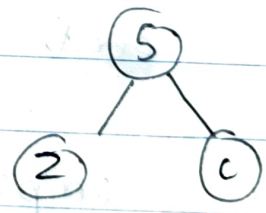
z c y x

②

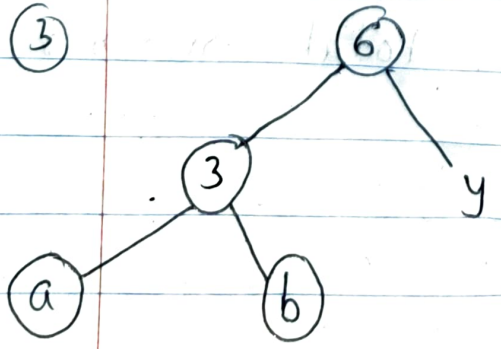


3 4

y x

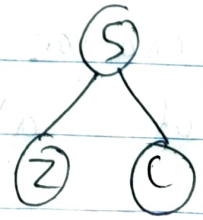


③

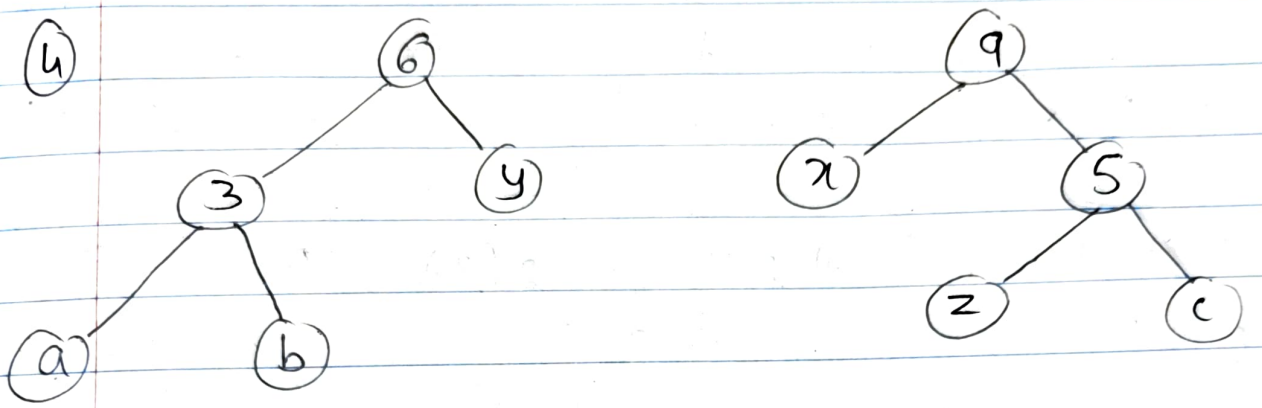


4

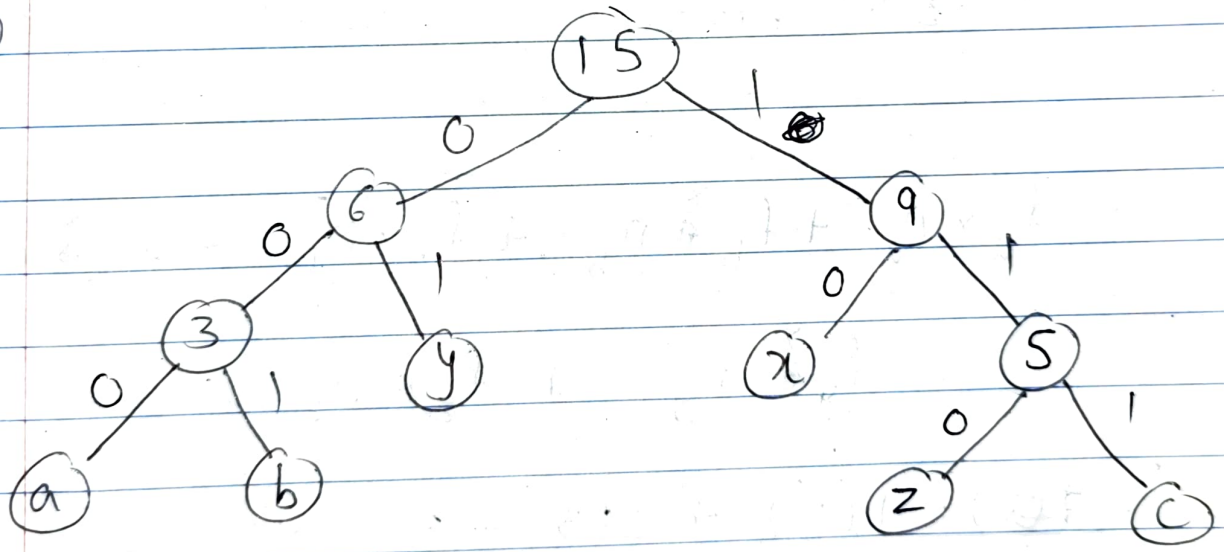
x



4



5



a = 000 ⇒ 3 bits

b = 001 ⇒ 3 bits

c = 111 ⇒ 3 bits

x = 10 ⇒ 2 bits

y = 01 ⇒ 2 bits

z = 110 ⇒ 3 bits

Bits needed to encode the string = frequency × no. of bits

= 1(3) + 2(3) + 2(3) + 4(2) + 3(2) + 2(3)

= 3 + 6 + 6 + 8 + 6 + 6

= 35 bits

$= f_a \times n_a + f_b \times n_b + f_c \times n_c + f_x \times n_x + f_y \times n_y + f_z \times n_z$

= 1(3) + 2(3) + 3(3) + 4(2) + 3(2) + 2(3)

= 3 + 6 + 9 + 8 + 6 + 6

= **38 bits**

We should also send the encoding and decoding logic with the encoded bit string.

∴ We send the ascii codes of each character and their corresponding encoded ~~bit~~ Huffman bit values together.

∴ Bits needed for ascii = 6 characters × 8
= 48

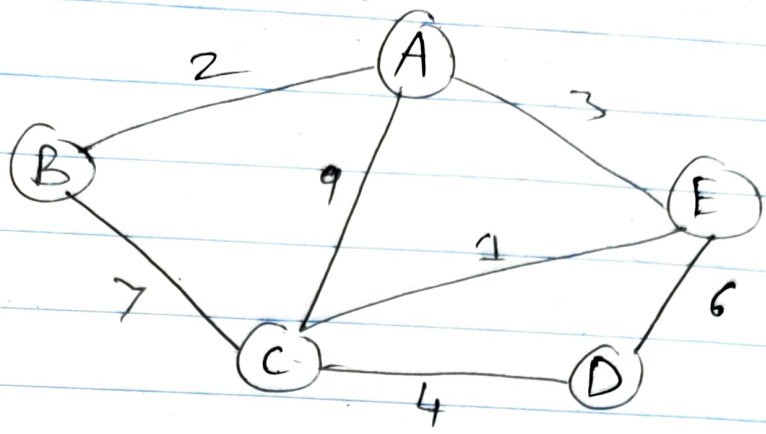
Also, the binary keys = no. of bits in (000, 001, 111, 10, 01, 110)
= 16

∴ Total bits which would be sent for future reference = 48 + 16 = 64

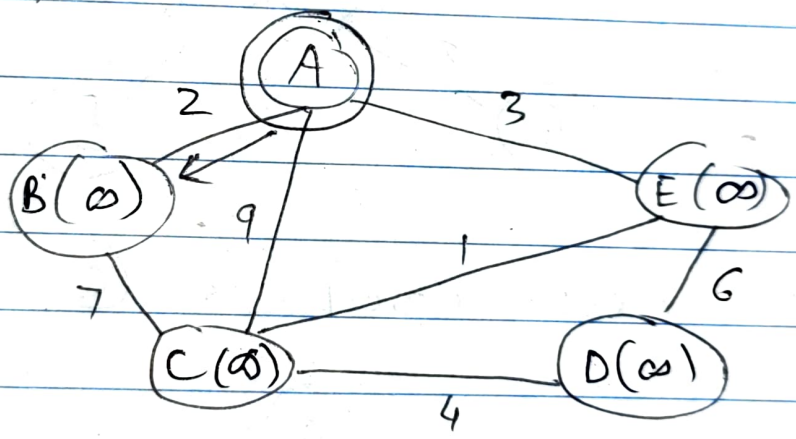
∴ Total bits with string = 64 + 38
= 102

Assignment 11

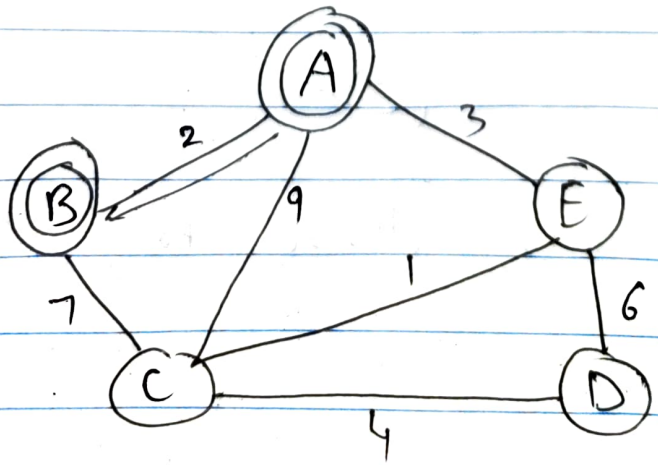
Q 3.



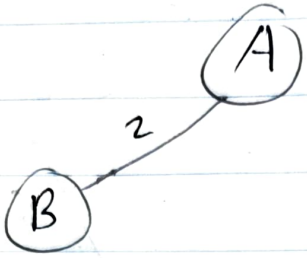
We start at vertex A, assume distance of all other vertices to be infinity for now. We have,



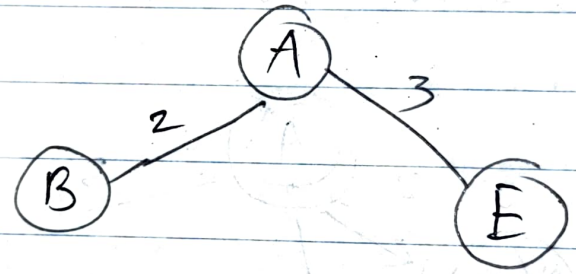
Now, we check nearest next vertex, i.e. B & add it to graph



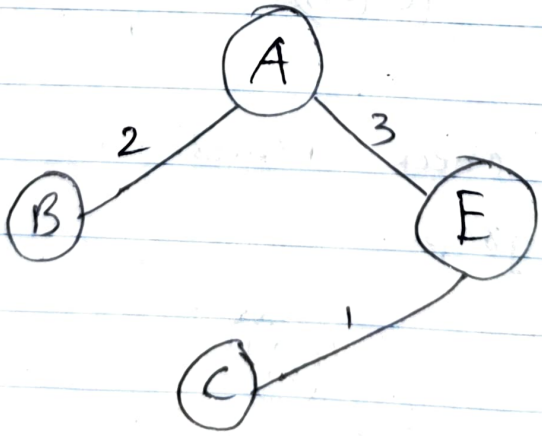
∴ We have the graph as



Next closest vertex from all the vertices in our graph is E.

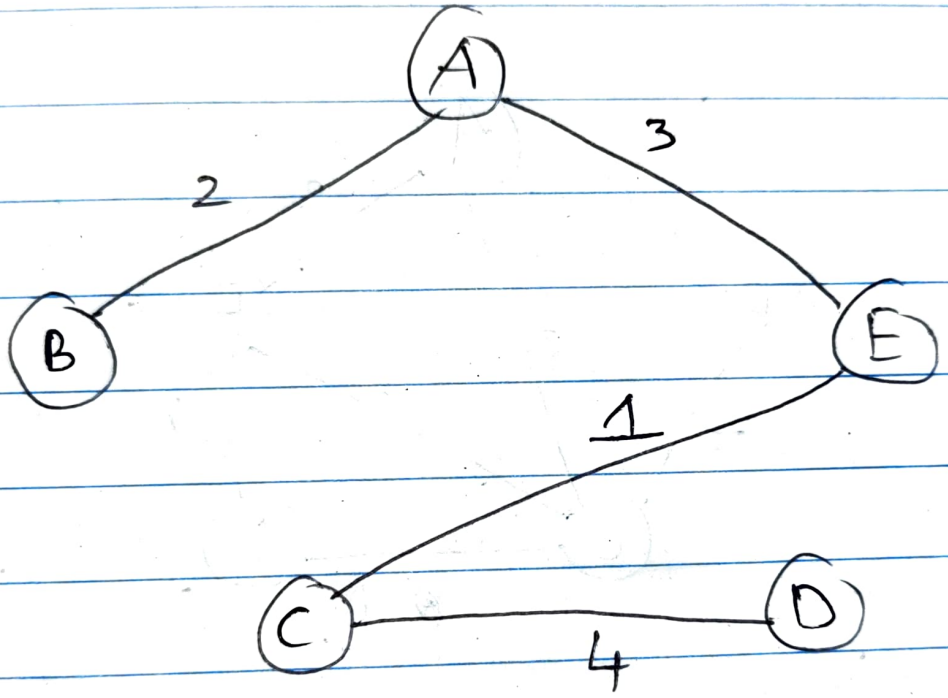


Now, next closest is (C) from (E)

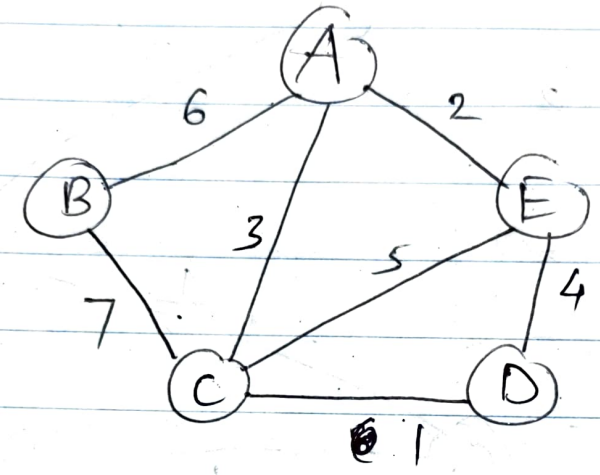


Now, (D) is closest from (C).

∴ Our final minimum spanning tree via Prim's algo →

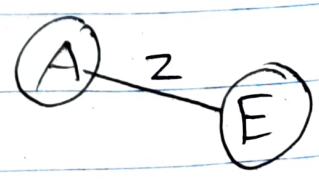


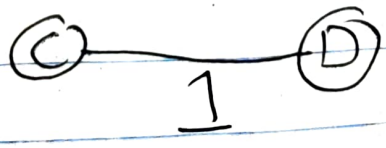
Q 4. Run Kruskal's algorithm on the below graph, show sequence of nodes / ~~node~~ edges.

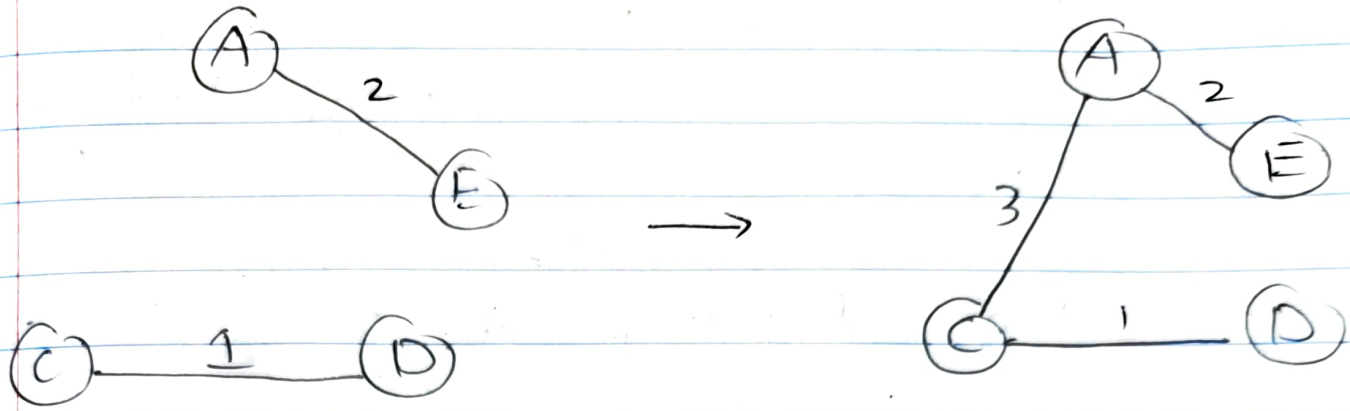


According to Kruskal's algo, we keep selecting the edges with the lowest cost in sequence as long as our resultant graph does not form a cycle. Skip all edges which result in a cycle.

∴ Shortest edge is C – D

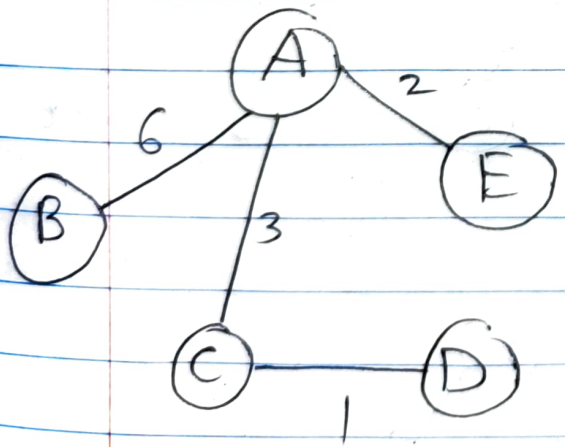






skip E-D because forms a cycle

skip C-E, because forms a cycle



∴ The sequence is C-D, then AE, then A-C and finally A-B